Chapter 2

Software engineering

Software engineering ethics

- Software engineering involves wider responsibilities than simply the application of technical skills.
- Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals.
- Ethical behaviour is more than simply upholding the law but involves following a set of principles that are morally correct.

Issues of professional responsibility

- Confidentiality
  - Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- Competence
  - Engineers should not misrepresent their level of competence. They should not knowingly accept work which is outwith their competence.
- Intellectual property rights
  - Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.
- Computer misuse

- Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses).

Rationale for the code of ethics

- *Computers have a central and growing role in commerce, industry, government, medicine, education, entertainment and society at large. Software engineers are those who contribute by direct participation or by teaching, to the analysis, specification, design, development, certification, maintenance and testing of software systems.*
- *Because of their roles in developing software systems, software engineers have significant opportunities to do good or cause harm, to enable others to do good or cause harm, or to influence others to do good or cause harm. To ensure, as much as possible, that their efforts will be used for good, software engineers must commit themselves to making software engineering a beneficial and respected profession.*

Ethical principles

1. PUBLIC - Software engineers shall act consistently with the public interest.

2. CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.

3. PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

4. JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.

5. MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

6. PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

7. COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.

8. SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Ethical dilemmas

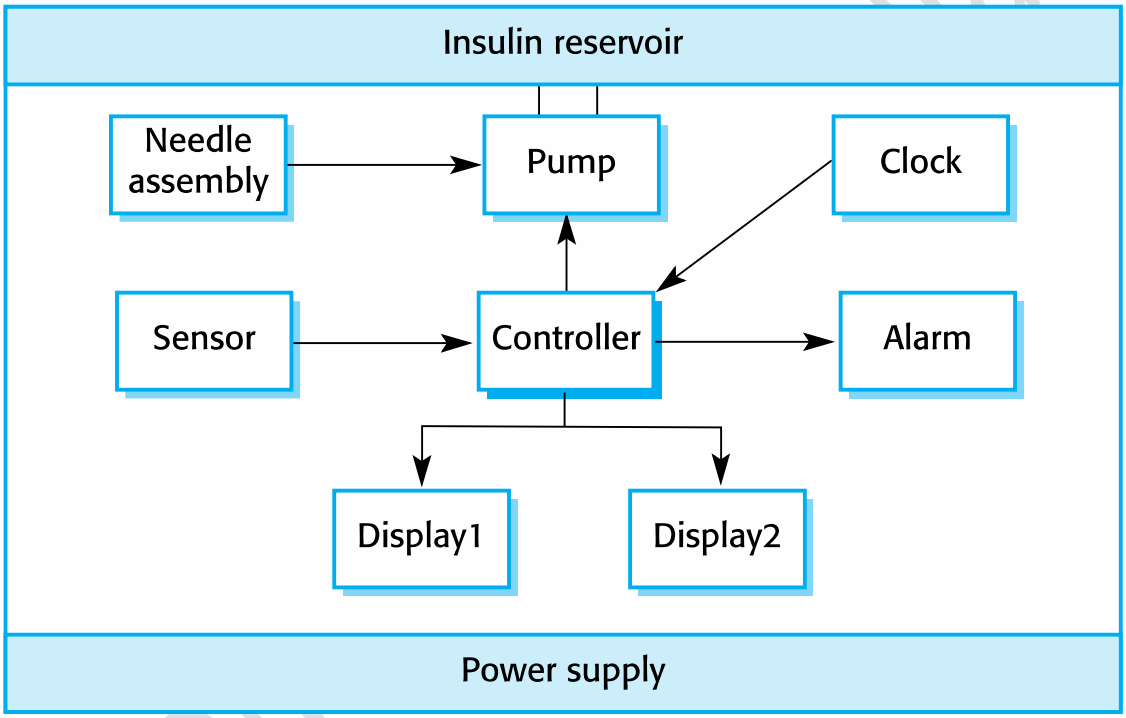
- Disagreement in principle with the policies of senior management.
- Your employer acts in an unethical way and releases a safety-critical system without finishing the testing of the system.
- Participation in the development of military weapons systems or nuclear systems.

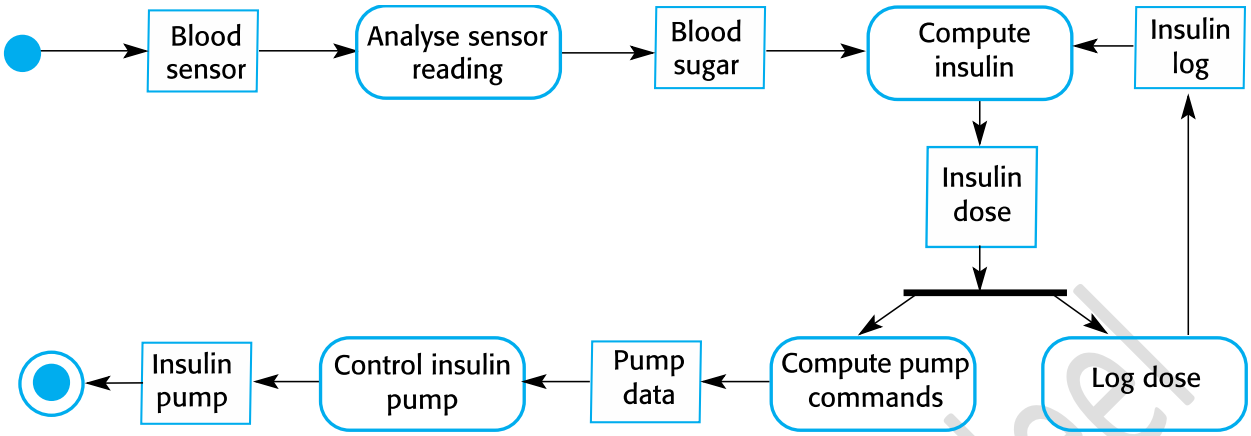
Case Studies

- A personal insulin pump
  - An embedded system in an insulin pump used by diabetics to maintain blood glucose control.
- A mental health case patient management system
  - Mentcare. A system used to maintain records of people receiving care for mental health problems.
- A wilderness weather station
  - A data collection system that collects data about weather conditions in remote areas.
- iLearn: a digital learning environment
  - A system to support learning in schools

Insulin pump control system

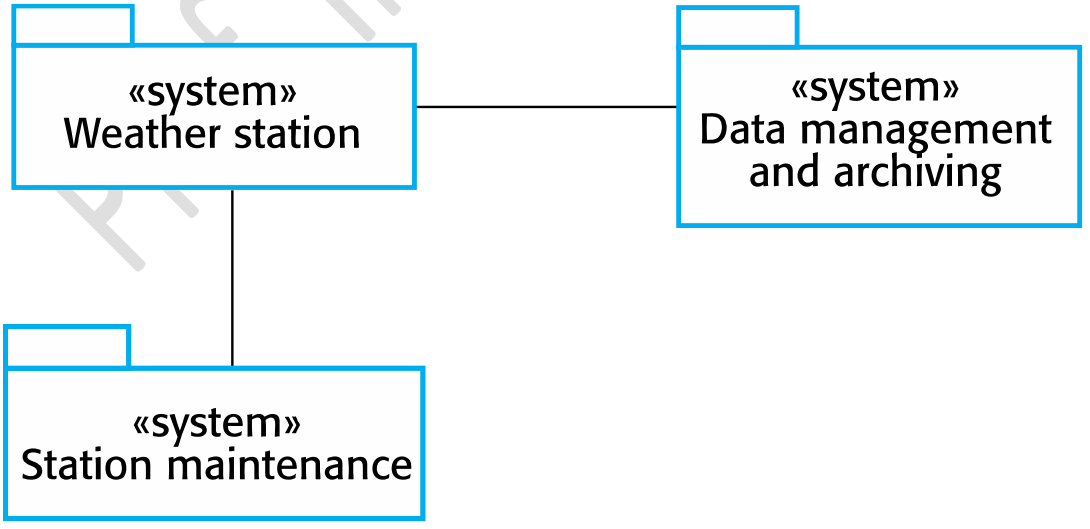
- ✧ Collects data from a blood sugar sensor and calculates the amount of insulin required to be injected.
- ✧ Calculation based on the rate of change of blood sugar levels.
- ✧ Sends signals to a micro-pump to deliver the correct dose of insulin.
- ✧ Safety-critical system as low blood sugars can lead to brain malfunctioning, coma and death; high-blood sugar levels have long-term consequences such as eye and kidney damage.





Wilderness weather station

- ✧ The government of a country with large areas of wilderness decides to deploy several hundred weather stations in remote areas.
- ✧ Weather stations collect data from a set of instruments that measure temperature and pressure, sunshine, rainfall, wind speed and wind direction.
  - The weather station includes a number of instruments that measure weather parameters such as the wind speed and direction, the ground and air temperatures, the barometric pressure and the rainfall over a 24-hour period. Each of these instruments is controlled by a software system that takes parameter readings periodically and manages the data collected from the instruments.



Weather information system

- The weather station system
  - This is responsible for collecting weather data, carrying out some initial data processing and transmitting it to the data management system.
- The data management and archiving system
  - This system collects the data from all of the wilderness weather stations, carries out data processing and analysis and archives the data.
- The station maintenance system
  - This system can communicate by satellite with all wilderness weather stations to monitor the health of these systems and provide reports of problems.

Additional software functionality

- Monitor the instruments, power and communication hardware and report faults to the management system.
- Manage the system power, ensuring that batteries are charged whenever the environmental conditions permit but also that generators are shut down in potentially damaging weather conditions, such as high wind.
- Support dynamic reconfiguration where parts of the software are replaced with new versions and where backup instruments are switched into the system in the event of system failure.

2.2 Software Life Cycle Models

A software lifecycle model is a series of steps through which the product progresses. These include requirements phase, specification phase, design phase, implementation phase, integration phase, maintenance phase, and retirement. Software Development Lifecycle Models illustrate the way you organize your activities. There are a number of Software Development Lifecycle Models, each having its strengths and weaknesses and suitable in different situations and project types.
The list of models includes the following:

2.2.1 Build and Fix Model

It is unfortunate that many products are developed using what is known as the build-and fix model. In this model the product is constructed without specification or any attempt at design. The developers simply build a product that is reworked as many times as necessary to satisfy the client. This model may work for small projects but is totally unsatisfactory for products of any reasonable size. The cost of build-and fix is actually far greater than the cost of properly specified and carefully designed product. Maintenance of the product can be extremely in the absence of any documentation. This model is illustrated in figure 2.1

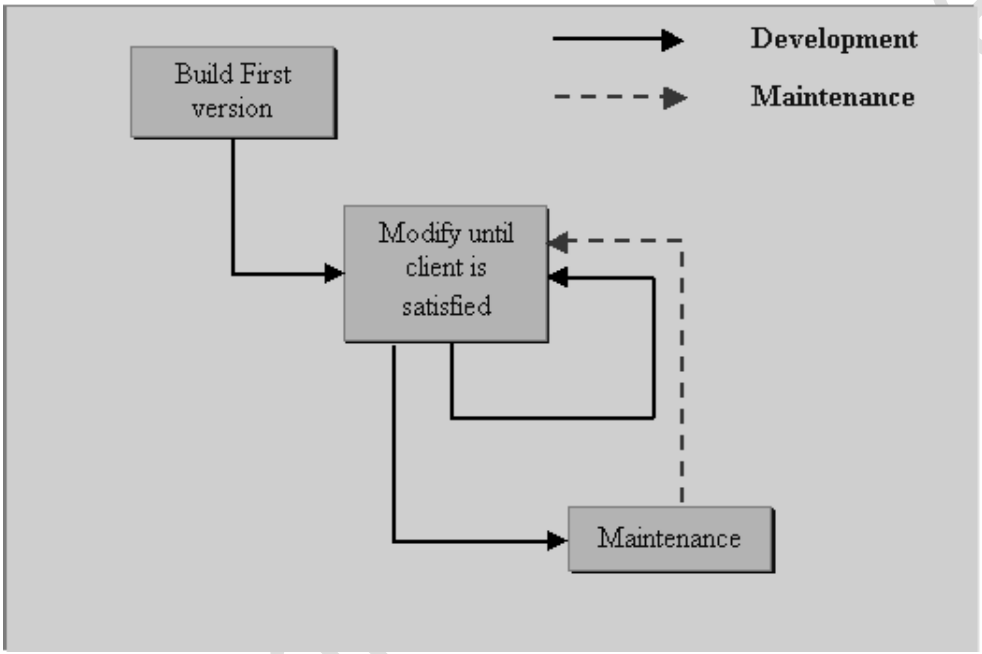


Figure 2.1: Build and fix model

2.2.2 Waterfall Model

The first published model of the software development process was derived from other engineering processes. Because of the cascade from one phase to another, this model is known as the waterfall model. It is also known as linear sequential model. This model is illustrated in figure 2.2.

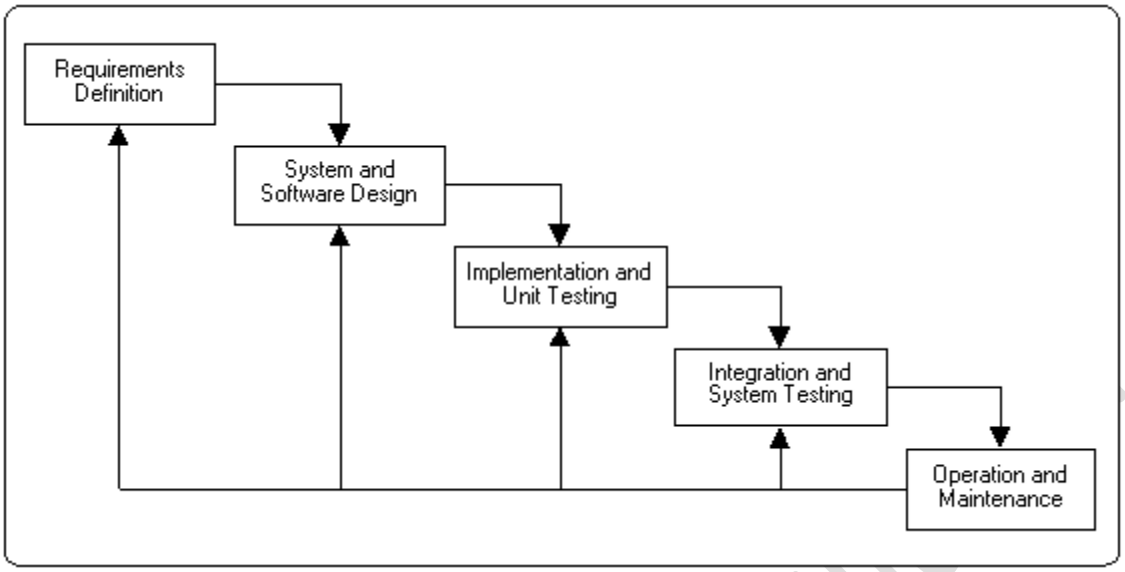


Figure 2.2 Waterfall model

It suggests a systematic, sequential approach to software development that begins at the system level and progresses through the analysis, design, coding, testing, and maintenance. The drawback of the waterfall model is the difficulty of accommodating change after the process is underway. The Waterfall model problems are:

- Inflexible partitioning of the project into distinct stages.
- This makes it difficult to respond to changing customer requirements.
- Therefore, this model is only appropriate when the requirements are well-understood.

2.2.3 The Prototyping Model

The Prototyping Model begins with requirements gathering. Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory. A "quick design" then occurs. The quick design focuses on a representation of those aspects of the software that will be visible to the customer/user (e.g., input approaches and output formats). The quick design leads to the construction of a prototype. This process is shown in figure 2.3. The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed. Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.

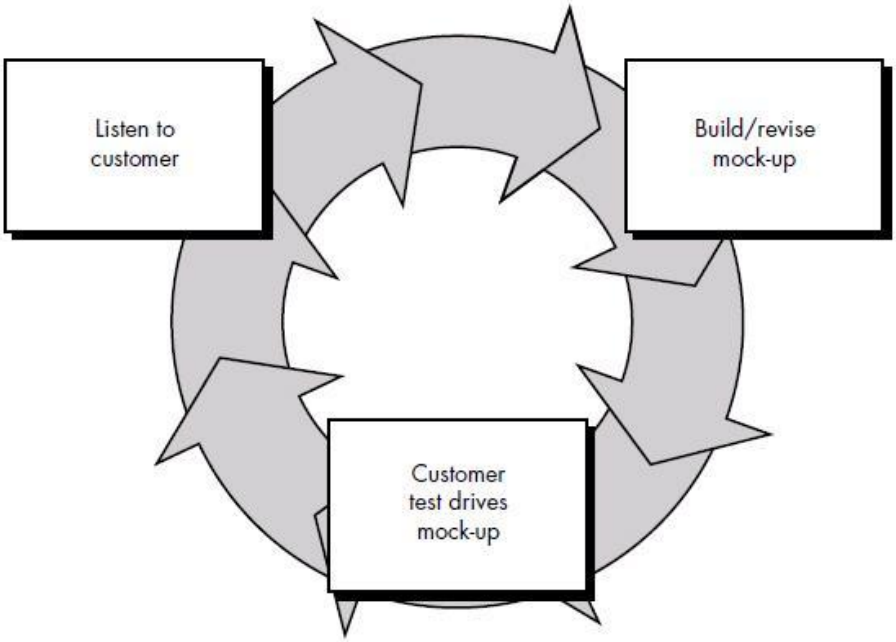


Figure 2.3 Prototyping Model

The Prototyping problems are:

1. Users treat the prototype as the solution.
2. The developer often makes implementation compromises in order to get a prototype working quickly.

2.2.4 The Incremental Model

The incremental model combines elements of the linear sequential model with the iterative philosophy of prototyping. Referring to Figure 2.4, the incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces a deliverable "increment" of the software. For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment. It should be noted that the process flow for any increment can incorporate the prototyping paradigm. When an incremental model is used, the first increment is often a core product. That is, basic requirements are addressed, but many supplementary features (some known, others unknown) remain undelivered. The core product is used by the customer. As a result of use and/or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of

additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced.

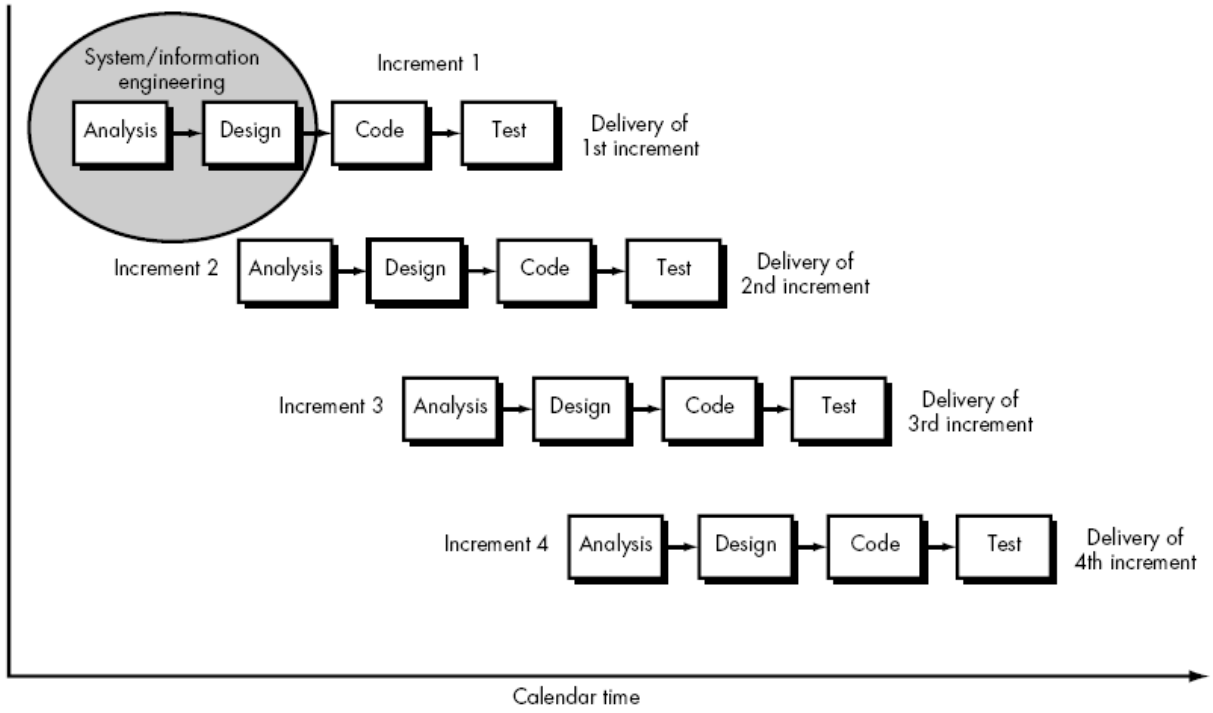


Figure 2.4 the Incremental Model

2.2.5 The Spiral Model

The spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model. It provides the potential for rapid development of incremental versions of the software. Using the spiral model software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype. During later iterations, increasingly more complete versions of the engineered system are produced. 'A spiral model is divided into a number of framework activities, also called task regions. Typically, there are between three and six task regions. Figure 2.5 depicts a spiral model that contains six task regions:

1. **Customer communication**: tasks required to establish effective communication between developer and customer.
2. **Planning**: tasks required to define resources, timelines, and other project related information.
3. **Risk analysis:** tasks required to assess both technical and management risks.
4. **Engineering**: tasks required to build one or more representations of the application.

5. **Construction and release**: tasks required to construct, test, install, and provide user support (e.g., documentation and training).
6. **Customer evaluation**: tasks required to obtain customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation stage.

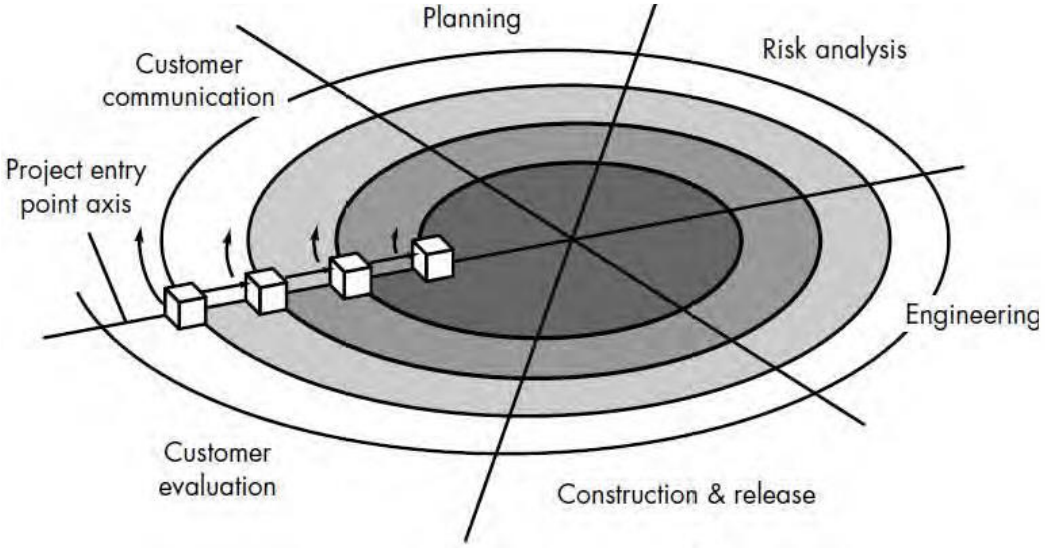


Figure 2.5 the Spiral Model

Each of the regions is populated by a set of work tasks, called a task set, that are adapted to the characteristics of the project to be undertaken. For small projects, the number of work tasks and their formality is low. For larger, more critical projects, each task region contains more work tasks that are defined to achieve a higher level of formality.

2.2.6 The Exploratory Model

The Exploratory- model, shown in figure 2.6, is extremely simple in its construction; it is composed of the following steps:

1. **Initial Specification Development:** using whatever information is immediately available, a brief system specification is created to provide a rudimentary starting point.
2. **System Construction / Modification (System Build):** a system is created and / or modified according to whatever information is available.
3. **System Testing:** the system is tested to see what it does, what can be learned from it, and how it may be improved.

4. **System Implementation (Delivery):** after many iterations of the previous two steps produce satisfactory results, the system is dubbed as "finished" and implemented.

This model is used in some situation in which it is very difficult, if not impossible, to identify any of the requirements for a system at the beginning of the project. Theoretical areas such as Artificial Intelligence are candidate for using the Exploratory Model, because much of the research in these areas is based on guess-work, estimation and hypothesis. In These cases, an assumption is made as to how the system might work and then rapid iterations are used to quickly incorporate suggested changes and build a usable system. A distinguishing characteristic of the Exploratory Model is the absence of precise specifications. Finishing project is based on the end result and not on its requirements.

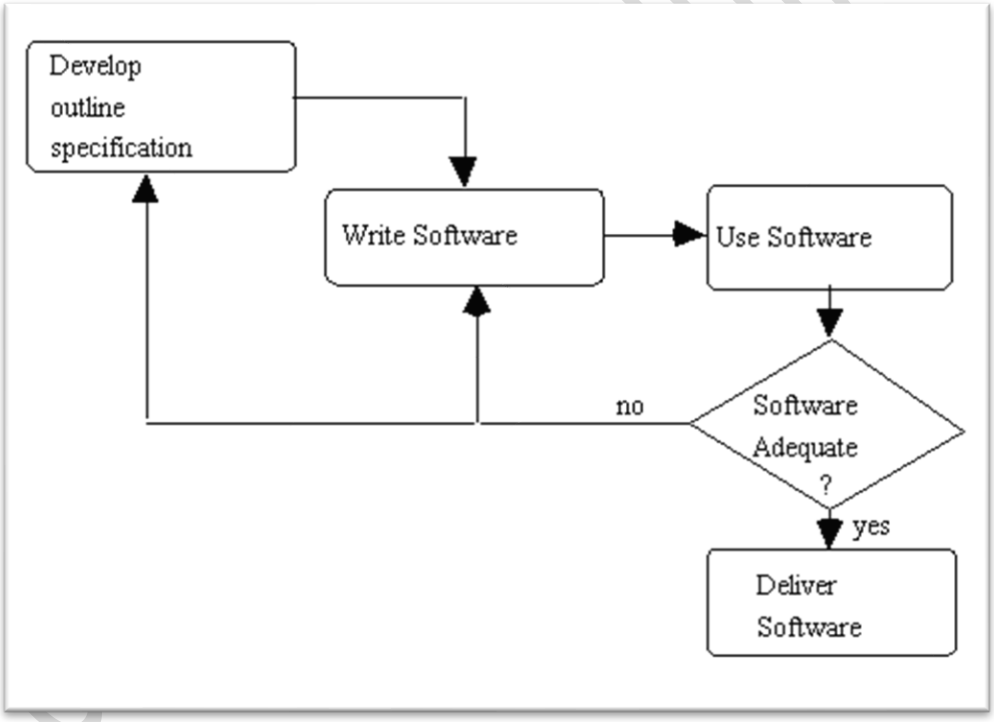


Figure 2.6 Exploratory Model